Compilerbau CS1019

Th. Letschert

TH Mittelhessen Gießen

University of Applied Sciences

Scanner / Parser: Positionsinfo, Fehlerbehandlung

- **Fehlerbehandlung**
- **Verwaltung von Positionsinformation**

Fehler in der Syntaxanalyse

Definition Fehler

Der Text entspricht nicht der Grammatik

Fehler-Entdeckung

- **In Scanner: Token im Text nicht erkennbar**
- **Im Parser: Token nicht Bestandteil einer möglichen Produktion**

Fehler-Behandlung

- **Panik:**
  Fehlermeldung und dann
  sofortiger Abbruch der Syntaxanalyse
- **Recovery (nur Parser):**
  Fehlermeldung, dann
  Wieder-Aufsetzen an einem späteren Punkt im Text
  (Behandeln wir hier nicht)

Fehler in der Syntaxanalyse

Fehlermeldung

- Ursache
- Position im Text an der der Fehler erkannt wurde

Positionen

Position

Position = Textposition

- Zeile
- Spalte

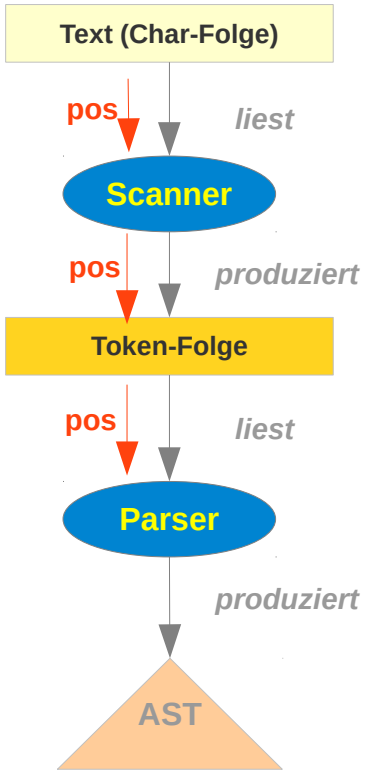
Fehler-Position

– Scanner: Position an der kein Token erkannt werden kann
– Parser: Position an der ein nicht erwartetes Token auftritt

Positions-Information

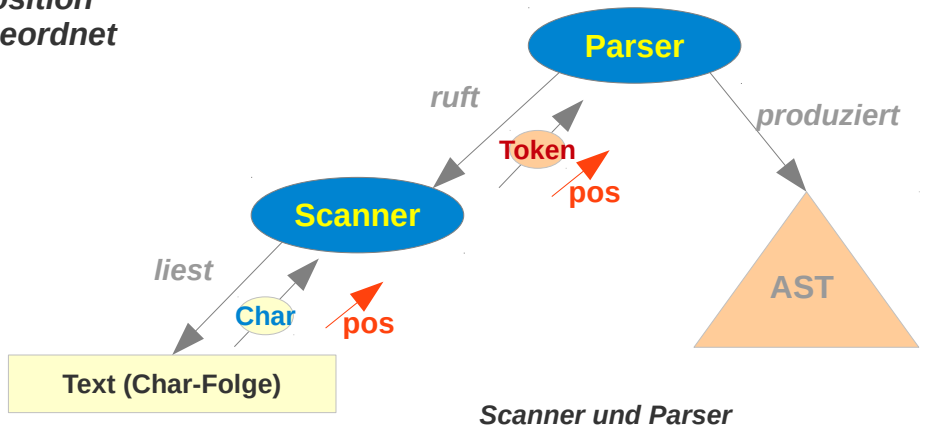
– Scanner benötigt
  Position(aktuelles Zeichen)
– Parser benötigt
  Position(aktuelles Token)

Positionsbehandlung



Zeichen ist eine Position zugeordnet

Tokens ist eine Position zugeordnet



Scanner und Parser operieren verschränkt

Positionsbehandlung – SW-Organisation

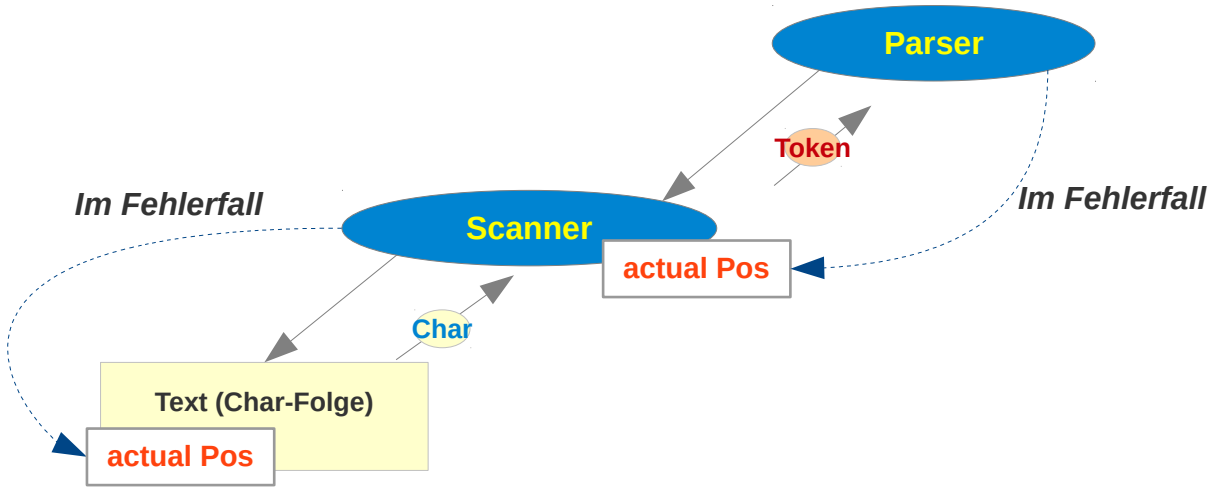
Die Positionsverwaltung kann auf verschiedene Arten organisiert werden, z.B.:

- **Objekte mit einer Postion beinhalten Positionsinfo**
- **Objekte die die Positionen ihrer Elemente kennen verwalten die Positionsinfo**

Positionsbehandlung – SW-Organisation

Objekte die die Positionen ihrer Elemente kennen verwalten die Positionsinfo

Im Fehlerfall wird darauf zugegriffen



Positionsbehandlung – SW-Organisation

Objekte mit einer Postion beinhalten Positionsinfo

Beispiel 1:

```
case class PositionedToken(t: Token, line: Int, col: Int) extends Token

. . .

val numberTokenWithPosition = PositionedToken(NumberToken("42"), 2, 3)
```

Plump und ungeeignet: Pattern-Match auf Tokenart ist nicht mehr direkt möglich: Alle Tokens sind erst mal Positionierte Tokens.

Beispiel 2:

```
abstract class WithPosition(line: Int, row:  Int)

abstract class Token(line: Int, col: Int) extends WithPosition(line, row)
case class  NumberToken(chars: String,line: Int, col:  Int)  extends Token(line, col)

. . .

val numberTokenWithPosition = NumberToken("42", 2, 3)
```

Pattern-Match auf Tokenart möglich, plumpe Lösung: keine „Separation of concerns“)

Positionsbehandlung – SW-Organisation

Objekte mit einer Postion beinhalten Positionsinfo

Beispiel 3 (*Separation of Concerns a la Scala*):

```
trait WithPosition {
  val line: Int
  val col:  Int
}

. . .

val numberTokenWithPosition: Token = new NumberToken("42") with WithPosition {
  val line = 2
  val col = 3
}
```

```
numberTokenWithPosition match {
  case t@NumberToken(x) =>
    println("Number token for number " + x)
    println(s"Token is at Position " +
      s"line: ${t.asInstanceOf[WithPosition].line}, " +
      s"col: ${t.asInstanceOf[WithPosition].col}"  )
}
```

oder

```
numberTokenWithPosition match {
  case t@NumberToken(x) =>
    println("Number token for number " + x)
    t match {
      case pos: WithPosition =>
        println(s"Token is at Pos " +
          s"line: ${pos.line}, " +
          s"row: ${pos.col}"  )
    }
}
```

Positionsbehandlung – SW-Organisation

Offset = Zeile / Spalte

```
trait WithPosition {
  val line: Int
  val col:  Int
}

object WithPosition {
  /**
    * compute line and row from a position in a string
    * @param s      the string
    * @param offset an offset within the string
    * @return       offset translated to line and column
    */
  def translateToPosition(s: String, offset: Int): (Int, Int) = {
    var lineNr = 0
    var colNr  = 0
    var pos = 0
    var lineStart = 0
    while (pos < offset) {
      if (s.charAt(pos) == '\n') {
        lineNr = lineNr + 1
        colNr = 0
      } else {
        colNr = colNr + 1
      }
      pos = pos + 1
    }
    (lineNr, colNr)
  }
}
```

Kann mit einem Cache für gefundene new-line-Zeichen beschleunigt werden.

Positionsbehandlung – Integration in den Scanner

```
...

private def nextToken(): Token = {
  import WithPosition._

  offset = skipWhiteSpace()

  actPos = offset

  var matched: Option[String] = None
  val startOffset = offset
  var i = 0

  while (! matched.isDefined && i < pats.length) { ... }

  matched match {
    case None => ...
    case Some(matchedStr) =>
      val (l, c) = translateToPosition(input, actPos)
      matchedStr match {
        case NumberPat(num)   => new NumberToken(num) with WithPosition { val line = l; val col = c }
        case AddOpPat(op)     => new AddOpToken(op) with WithPosition { val line = l; val col = c }
        case MultOpPat(op)    => new MultOpToken(op) with WithPosition { val line = l; val col = c }
        case LeftPPat(p)      => new LeftPToken(p) with WithPosition { val line = l; val col = c }
        case RightPPat(p)     => new RightPToken(p) with WithPosition { val line = l; val col = c }
        case x                => new ErrorToken("unexpected "+x) with WithPosition { val line = l; val col = c }
      }
  }
}

private var actToken: Token = nextToken()
private var actPos: Int     = 0

...
```